

Module: Systèmes Embarqués

Chapitre 2:
Les Processeurs

El Kefi HLEL

Janvier 2014

DÉFINITION D'UN PROCESSEUR

- Le processeur (CPU: Central Processing Unit) est un composant qui exécute des instructions séquentiellement (programme) a partir de données
- Il possède généralement : Unité de calcul (UAL) + Unité de commande + Registres + Horloge + Mémoire cache



















Composition d'un Processeur

Composition d'un Processeur (1)

Les parties essentielles d'un processeur sont :

- **Unité Arithmétique et Logique** (UAL): prend en charge les calculs arithmétiques et logiques élémentaires
- **Unité de contrôle**: Unité qui coordonne le fonctionnement des autres éléments dans le but d'exécuter une séquence d'instructions. Constituée de plusieurs éléments :
  - Registre d'Instruction (RI) : reçoit le code de la prochaine instruction à exécuter
  - Décodeur : détermine l'opération à exécuter à partir du code de l'instruction
  - Horloge : permet de synchroniser les différents éléments du processeur
  - Compteur Ordinal (CO) : registre contenant l'adresse du mot mémoire stockant le code de la prochaine instruction
  - Séquenceur : Automate générant les signaux de commande contrôlant les différentes unités. Il existe 2 façons de réaliser cet automate : séquenceur câblé ou un séquenceur microprogramme

COMPOSITION D'UN PROCESSEUR (2)

- **Registres**: sont des mémoires de petite taille (quelques octets). L'UAL est capable de manipuler leurs contenus à chaque cycle de l'horloge
- **Horloge**
  - Synchronise toutes les actions d'un système embarqué
  - Présente uniquement dans les processeurs synchrones
  - Définit le cycle de base : cycle machine
  - Le temps d'exécution d'une instruction n'est généralement pas égal au cycle machine, car il faut plusieurs cycles machine pour pouvoir séquentiellement récupérer l'instruction, la décoder et récupérer les données qu'elle doit traiter
  - Exemple: Sur les microcontrôleurs PIC, il faut 4 cycles d'horloge (machine) pour exécuter une instruction
- **Unité d'entrée-sortie**
  - Prend en charge la communication avec la mémoire de système
  - Assure la transmission des ordres destinés à piloter quelques processeurs spécialisés
  - Permettant au processeur d'accéder aux périphériques de système

Composition d'un Processeur: Autres Eléments

Les processeurs actuels intègrent également des éléments plus complexes :

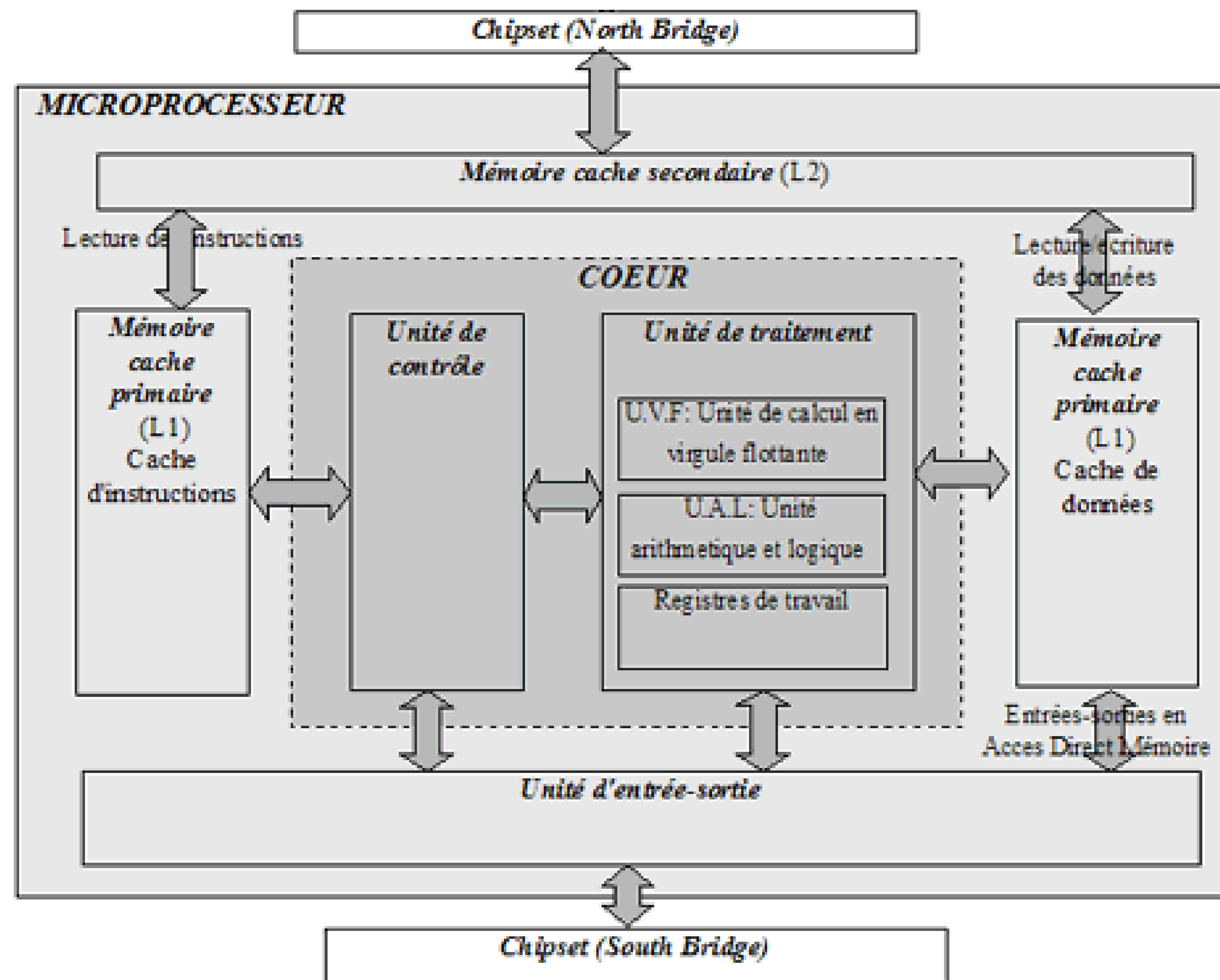
- **Plusieurs UAL**: Ce qui permet de traiter plusieurs instructions en même temps. Chaque UAL pouvant exécuter une instruction indépendamment de l'autre
- **Un pipeline** permet de découper temporellement les traitements à éffectuer
- **Une unité de calcul en virgule flottante** (*Floating Point Unit - FPU*), qui permet d'accélérer les calculs sur des nombres réels codés en virgule flottante (selon le format défini par IEEE754)
- **La mémoire cache**
  - Permet d'accélérer les traitements en diminuant les temps d'accès à la mémoire
  - Ces mémoires sont beaucoup plus rapides que la RAM et ralentissent moins le CPU
  - Le cache instructions reçoit les prochaines instructions à exécuter, le cache données manipule les données
  - Parfois, un seul cache est utilisé pour le code et les données
  - Plusieurs niveaux de caches peuvent coexister, on les désigne souvent sous les noms de L1, L2, L3 ou L4

Les Registres

Un certain nombre de registres sont communs à la plupart des processeurs:

- Registres de fonctionnement
  - Compteur Ordinal (CO), Registre Instruction (RI)…
  - Accumulateur (utilisé pour stocker les données en cours de traitement par l'UAL)
- Registres généraux
  - Servent à stocker des valeurs souvent utilisées ou des résultats intermédiaires sans passer par la mémoire
- Registres de pile (SP : Stack Pointer)
- Registres d'état (PSW : Program Status Word)
  - Ensemble de bits représentant chacun un état particulier (drapeau ou flag)
  - C : dépassement de capacité après un calcul de l'ALU
  - Z : résultat de l'opération est égal a 0

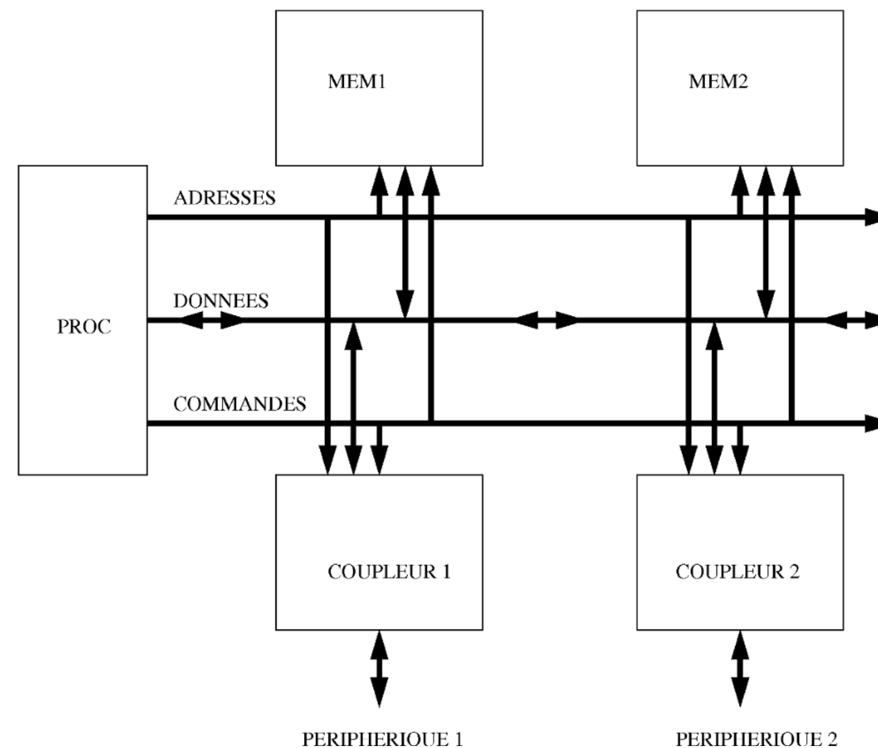
Exemple d'un Processeur



TYPES DE BUS

Un processeur possède trois types de bus :

- Un **bus de données**: définit la taille des données pour les entrées/sorties, dont les accès à la mémoire
- Un **bus d'adresse** permet, lors d'une lecture ou une écriture, d'envoyer l'adresse où elle s'effectue
- Un **bus de contrôle** permet la gestion du matériel, via les interruptions



Architecture, Classification & Opérations d'un Processeur

Classification des Processeurs

Un processeur est défini par :

- Son architecture
  - Son jeu d'instructions (*ISA*, *Instructions Set Architecture*)
  - La largeur de ses registres internes de manipulation de données (8, 16, 32, 64, 128) bits et leurs utilisations
  - Les spécifications des entrées/sorties, de l'accès à la mémoire, etc.
- Ses caractéristiques
  - La cadence de son horloge exprimée en MHz ou GHz
  - Sa finesse de gravure exprimée en nm (nanomètres) – la technologie de fabrication
  - Le nombre de noyaux de calcul (cœurs)

Classification des Architectures

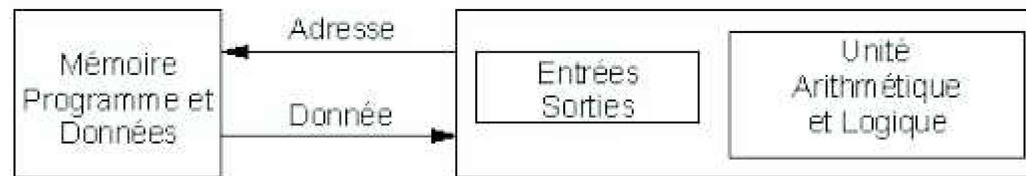
On classe les architectures en plusieurs grandes familles :

- CISC (Complex Instruction Set Computer): Choix d'instructions aussi proches que possible d'un langage de haut niveau. Le jeu d'instructions comporte souvent de nombreuses instructions complexes qui seront réalisées en plusieurs cycles
  - Exemples : x86 (Intel et AMD), Motorola 680x0
- RISC (Reduced Instruction Set Computer): Choix d'instructions plus simples. Chacune de ces instructions est censée être exécutée en un seul cycle. L'architecture propose en général un nombre important de registres généraux
  - Exemples : SPARC, PowerPC, MIPS, ARM
- Architecture VLIW (Very Long Instruction Word): Ce concept fait reposer une partie de la gestion du pipeline sur le compilateur : le processeur reçoit des instructions longues qui regroupent plusieurs instructions plus simples
- DSP (Digital Signal Processor): sont des processeurs spécialisés pour les calculs liés au traitement de signaux
- Un processeur softcore est un « circuit logique programmable » et n'a plus du tout de fonction pré-câblée

Type d'Architecture: Harvard Vs Von Neuman

Liaison CPU / Mémoire (Programme / Données)

- La suite d'instructions à exécuter est contenue dans une partie de la mémoire nommée mémoire-programme. Les données sur lesquelles opère le processeur sont stockées dans la partie mémoire-données.





LES OPÉRATIONS DU PROCESSEUR

- Le rôle fondamental de la plupart des processeurs est d'exécuter une série d'instructions stockées appelées « programme »
- Les instructions et les données transmises au processeur sont exprimées en mots binaires (code machine). Elles sont généralement stockées dans la mémoire. Le séquenceur ordonne la lecture du contenu de la mémoire
- Le langage le plus proche du code machine appelé langage assembleur. Toutefois, l'informatique à développé toute une série de langages, dits de « haut niveau » (comme le Pascal, C, C++, Fortran, Ada, etc), destinés à simplifier l'écriture des programmes
- Le programme est représenté par une série d'instructions qui réalisent des opérations en liaison avec la mémoire vive (RAM). Il y a quatre étapes que presque toutes les architectures utilisent :
  - fetch - recherche de l'instruction
  - decode - décodage de l'instruction (opération et opérandes)
  - execute - exécution de l'opération
  - writeback - écriture du résultat

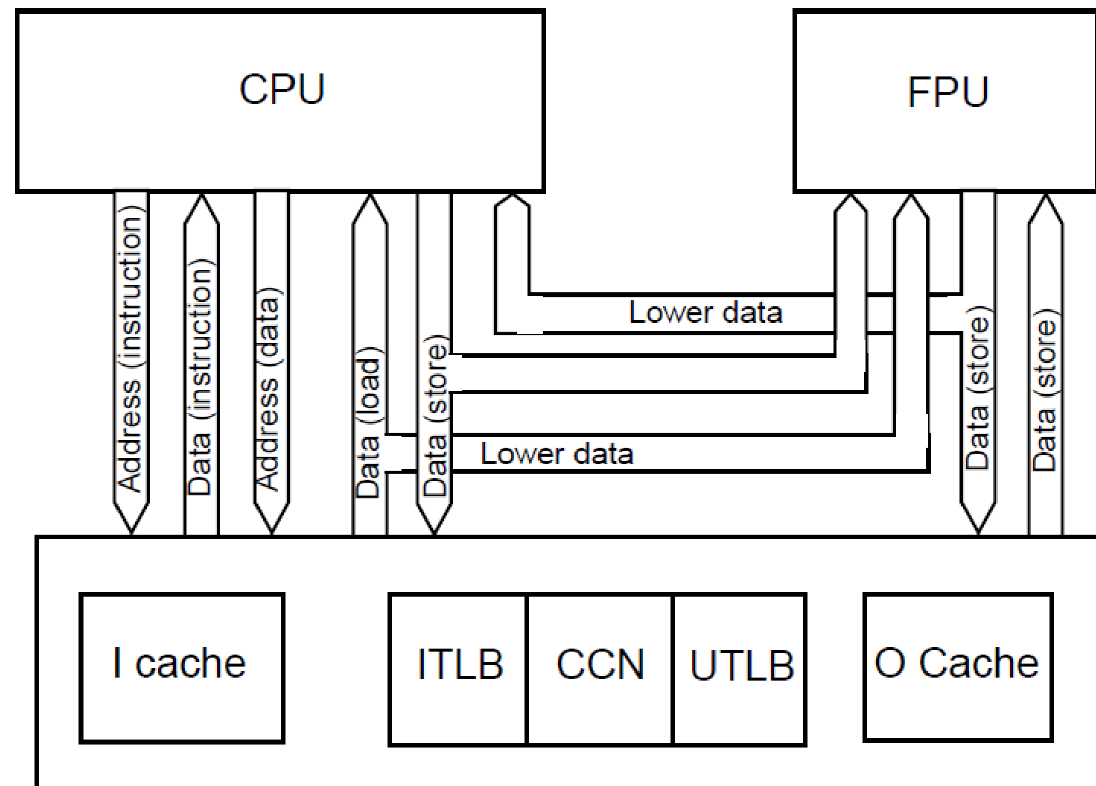
VITESSE DE TRAITEMENT

- La vitesse de traitement d'un processeur est exprimée en MIPS (million d'instructions par seconde) ou en mégaFLOPS (millions de floating-point opérations per second) pour la partie virgule flottante (FPU)
- Les processeurs sont basés sur différentes architectures et techniques de parallélisation des traitements qui ne permettent plus de déterminer simplement leurs performances
- Des programmes spécifiques d'évaluation des performances (benchmarks) ont été mis au point pour obtenir des comparatifs des temps d'exécution de programmes réels: dhrystone, cyclesoak, lmbench, …

Exemple: Le Processeur SH-4

LE CORE SH4: SCHÉMA

Le **SuperH** est une famille de processeurs, conçu à la base par Hitachi. L'architecture des SuperH est de type RISC. Ils sont principalement utilisés dans les systèmes embarqués



Les registres de SH-4

Type	Registers	Initial value[a]
General registers	R0_BANK0–R7_BANK0, R0_BANK1–R7_BAN K1, R8–R15	Undefined
Control registers	SR	MD bit = 1, RB bit = 1, BL bit = 1, FD bit = 0, I3–I0 = 1111 (0xF), reserved bits = 0, others undefined
	GBR, SSR, SPC, SGR, DBR	Undefined
	VBR	0x00000000
System registers	MACH, MACL, PR, FPUL	Undefined
	PC	0xA0000000
	FPSCR	0x00040001
Floating-point reg-isters	FR0–FR15, XF0–XF15	Undefined

MÉMOIRES CACHES DE SH-4

Item	Instruction cache	Operand cache
Capacity	8-kbyte cache	16-kbyte cache or 8-kbyte cache + 8-kbyte RAM
Type	Direct mapping	Direct mapping
Line size	32 bytes	32 bytes

Name	Abbreviation	R/W	Initial value[a]	P4 address[b]	Area 7 address[b]	Access size
Cache control register	CCR	R/W	0x0000 0000	0xFF00 001C	0x1F00 001C	32
Queue address control register 0	QACR0	R/W	Undefined	0xFF00 0038	0x1F00 0038	32
Queue address control register 1	QACR1	R/W	Undefined	0xFF00 003C	0x1F00 003C	32

SH-4: Jeu d'Instructions

Instruction:

- Opération élémentaire qu'un programme demande à un processeur d'effectuer
- Les instructions sont codées en binaire. Un champ de l'instruction appelé code opération ou **opcode** désigne l'opération à éffectuer
- Puisque sa valeur numérique n'a pas de sens, le programmeur utilise une abréviation désignant le code opération fourni par le langage assembleur pour ce processeur

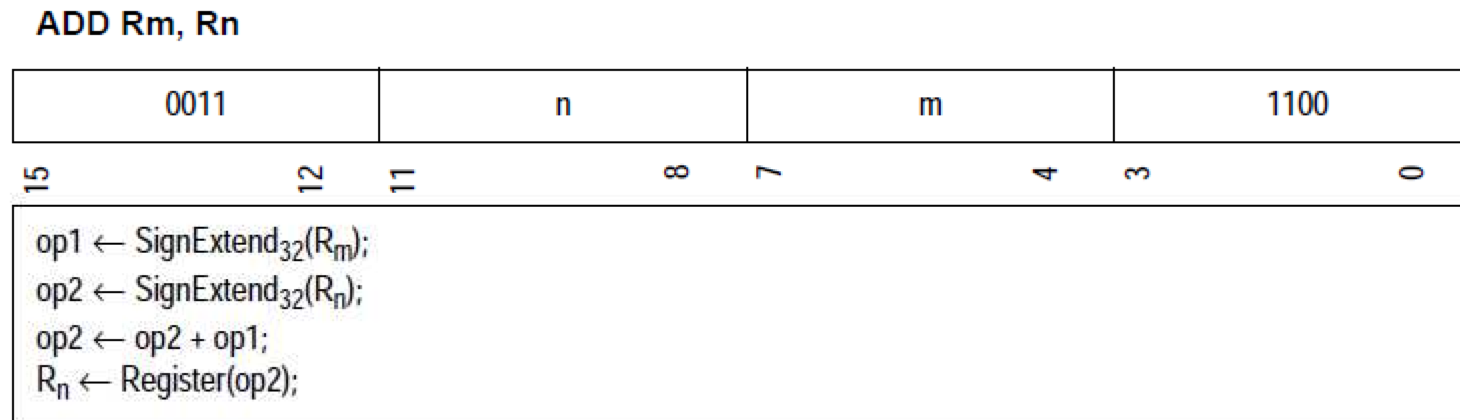
Jeu d'instructions (Sh4) [instruction de taille 2 bytes]:

- Transfert: 39 instructions (MOV, SWAP, …)
- Arithmétique: 33 instructions (ADD, CMP, DIV, …)
- Logique: 14 instructions (AND, NOT, XOR, …)
- Décalage: 16 instructions
- Branchement: 11 instructions
- Contrôle de système: 57 instructions
- Virgule flottante (simple précision): 30 instructions
- Virgule flottante (double précision): 13 instructions
- Virgule flottante (Contrôle): 8 instructions

EXEMPLES D'INSTRUCTIONS (1)

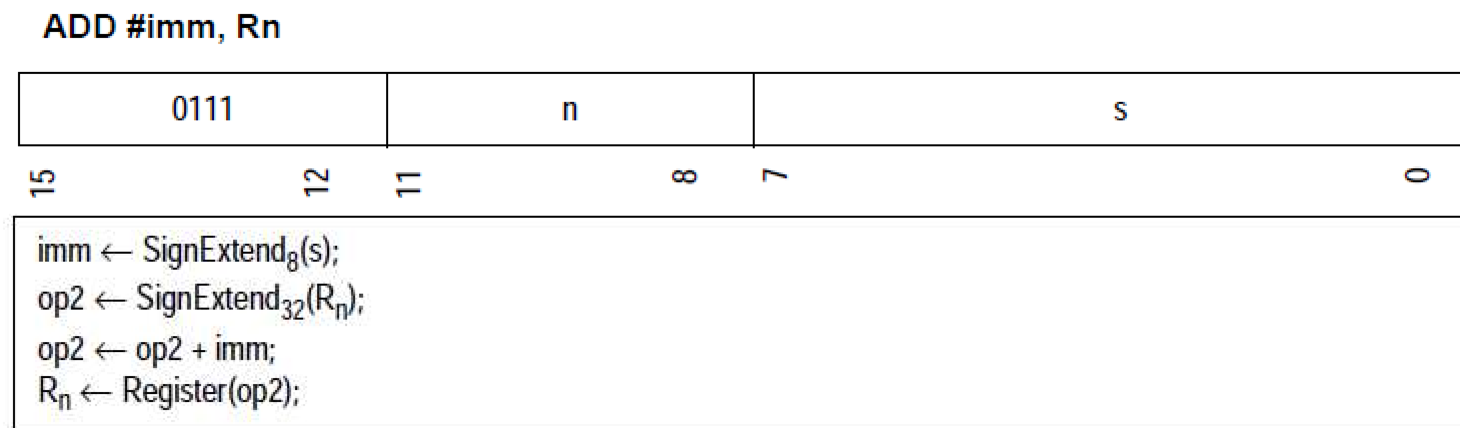
- **ADD Rm, Rn**

Faire l'addition des 2 registres Rm et Rn et placer le résultat dans le registre Rn



- **ADD #imm, Rn**

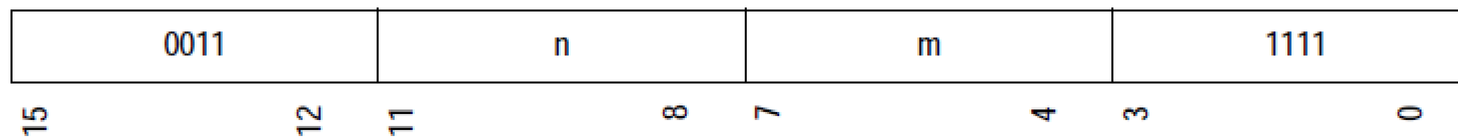
Faire l'addition du registre Rn et l'octet (s) et placer le résultat dans le registre Rn



- **ADDV Rm, Rn**

This instruction adds Rm to Rn and places the result in Rn. The T-bit is set to 1 if the addition result is outside the 32-bit signed range, otherwise the T-bit is set to 0.

ADDV Rm, Rn



$op1 \leftarrow SignExtend_{32}(R_m);$

$op2 \leftarrow SignExtend_{32}(R_n);$

$op2 \leftarrow op2 + op1;$

$t \leftarrow INT\ ((op2 < (-\ 2^{31}))\ OR\ (op2 \geq 2^{31}));$

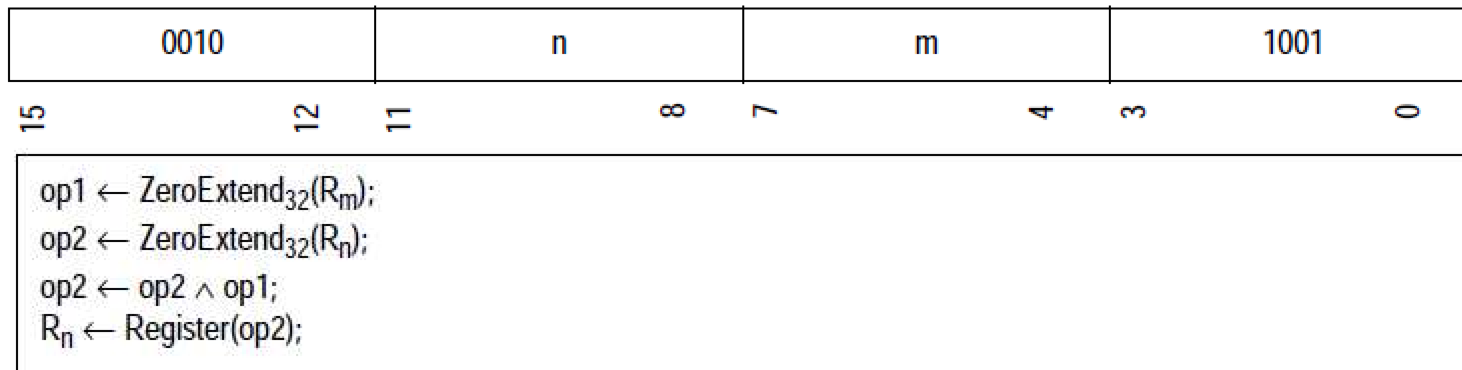
$R_n \leftarrow Register(op2);$

$T \leftarrow Bit(t);$

EXEMPLES D'INSTRUCTIONS (3)

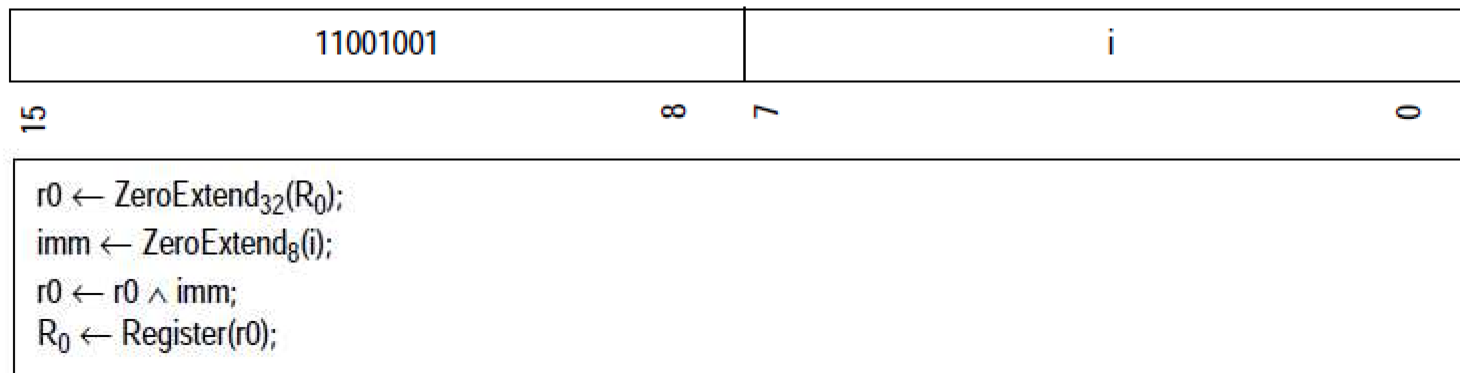
- **AND Rm, Rn**

This instruction performs bitwise AND of Rm with Rn and places the result in Rn.



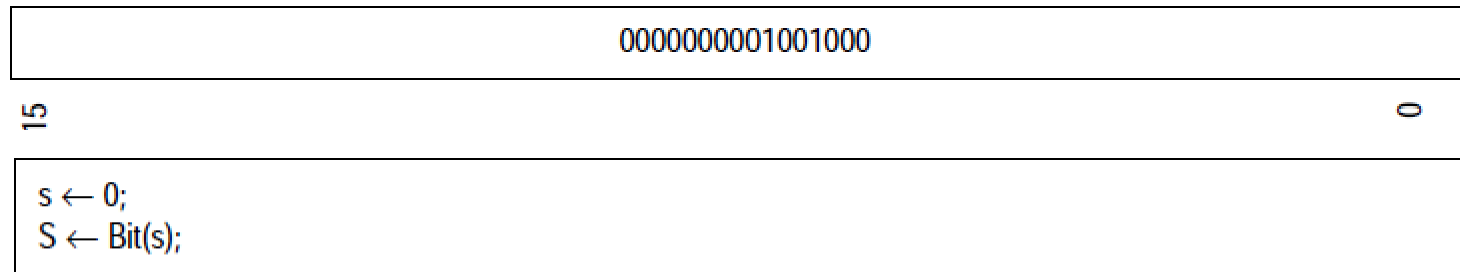
- **AND #imm, R0**

This instruction performs bitwise AND of R0 with the zero-extended 8-bit immediate i and places the result in R0.



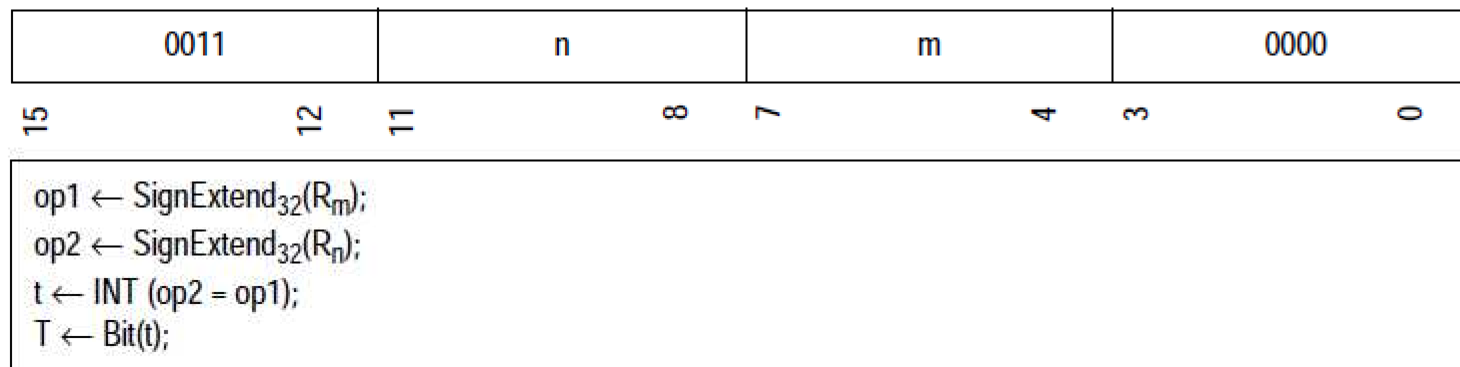
- **CLRS**

This instruction clears the S-bit.



- **CMP/EQ Rm, Rn**

This instruction sets the T-bit if the value of Rn is equal to the value of Rm, otherwise it clears the T-bit.

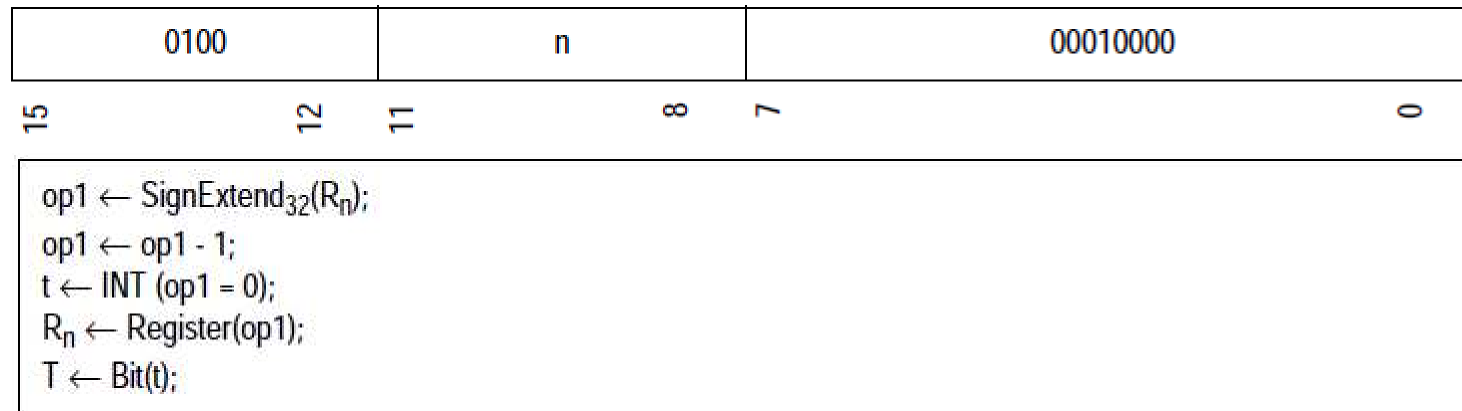


EXEMPLES D'INSTRUCTIONS (5)

- **DT Rn**

This instruction subtracts 1 from Rn and placed the result in Rn. The T-bit is set if the result is zero, otherwise the T-bit is cleared.

DT Rn



15–12	11–8	7–0
0100	n	00010000

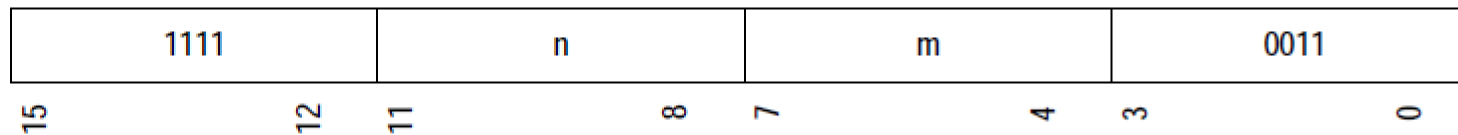
```
op1 ← SignExtend32(Rn);
op1 ← op1 - 1;
t ← INT (op1 = 0);
Rn ← Register(op1);
T ← Bit(t);
```

EXEMPLES D'INSTRUCTIONS (6)

- **FDIV FRm, FRn**

This floating-point instruction performs a single-precision floating-point division. It divides FRn by FRm and places the result in FRn. The rounding mode is determined by FPSCR.RM.

FDIV FRm, FRn




```
Available only when PR=0
```

```
sr ← ZeroExtend32(SR);
fps ← ZeroExtend32(FPSCR);
op1 ← FloatValue32(FRm);
op2 ← FloatValue32(FRn);
IF (FpuIsDisabled(sr) AND IsDelaySlot())
    THROW SLOTFPUDIS;
IF (FpuIsDisabled(sr))
    THROW FPUDIS;
op2, fps ← FDIV_S(op2, op1, fps);
IF (FpuEnableV(fps) AND FpuCauseV(fps))
    THROW FPUEXC, fps;
IF (FpuEnableZ(fps) AND FpuCauseZ(fps))
    THROW FPUEXC, fps;
IF (FpuCauseE(fps))
    THROW FPUEXC, fps;
IF ((FpuEnableI(fps) OR FpuEnableO(fps)) OR FpuEnableU(fps))
    THROW FPUEXC, fps;
FRn ← FloatRegister32(op2);
FPSCR ← ZeroExtend32(fps);
```